UNIVERSITY OF CALIFORNIA
College of Engineering
Department of Electrical Engineering and Computer Sciences
Last modified on October 17, 2012.

Elad Alon ***FALL 2012 TERM PROJECT PHASE I*** *EECS 141*

1. Design of a SLICEL for an FPGA – Background

Field Programmable Gate Arrays (FPGA) are digital integrated circuits whose functionality can be reconfigured after manufacturing. Although their performance in terms of delay, power, and area are worse than a customized Application Specific Integrated Circuit (ASIC), FPGAs are gaining increasing adoption because of their fast prototyping capabilities and because of their flexibility (one die can be used for many different designs, while in an ASIC implementation a new specialized chip is redesigned for every application). Since both these factors reduce the overall cost of a project, FPGAs are often the chosen architecture.

FPGAs are modular structures made up of reconfigurable logic blocks and interconnects. Both the functionality of the blocks and the connections between them are programmable after manufacturing. The logic blocks are often organized in *slices*, i.e., atomic units with defined connections to the interconnects and to the other computational blocks. Within the slice, in order to support programmability, logic typically is not implemented using traditional gates (e.g. NAND, NOR, etc.). Instead, the truth table of the logic function to be implemented is stored in a Look-Up Table (LUT). The inputs of the function become the address input of the LUT cell where the corresponding value of the output is stored. This structure adds flexibility to the FPGA, since the values stored in the LUT can be programmed differently for each design.

A typical implementation of a LUT uses an SRAM. Since the number of inputs and outputs of the function to be programmed inside a slice is not known a priori, the SRAM LUT is partitioned into sub-blocks, and additional logic (based on MUXs) is then used to select the desired input/output combinations. For example, two 5-input LUTs can implement either two 5-input functions or one 6-input function, where the 6^{th} input is the selection bit of a MUX connected to the outputs of the two LUTs.

Finally, the slice is often augmented with custom logic to enhance the performance (delay, power) of commonly executed operations. This custom logic is particularly important to speed up operations that require interactions between the outputs of multiple LUTs - e.g., additions.

In this project, you will design a type of slice known as a SLICEL, including four LUTs with reconfigurable numbers of inputs and outputs (set via MUXs), and additional "vertical" logic to implement an adder. In the first stage of this project, you will be characterizing the SRAM cell provided to you and designing the critical path of the address decoder for the SRAM.

1.1 High level structure

A block diagram of the SLICEL you will be designing is shown in Figure 1. There are three major blocks to be designed:

- Four 5/6-input LUTs, whose inputs are labeled A[5:0], B[5:0], C[5:0] and D[5:0]..
- MUX-based logic, to select the desired configuration of inputs/outputs. Three more inputs (AX, BX and CX) are added to the MUXs to allow operating the slice as two 7-input LUTs or one 8-input LUT.
- Additional vertical logic supporting the implementation of an adder.

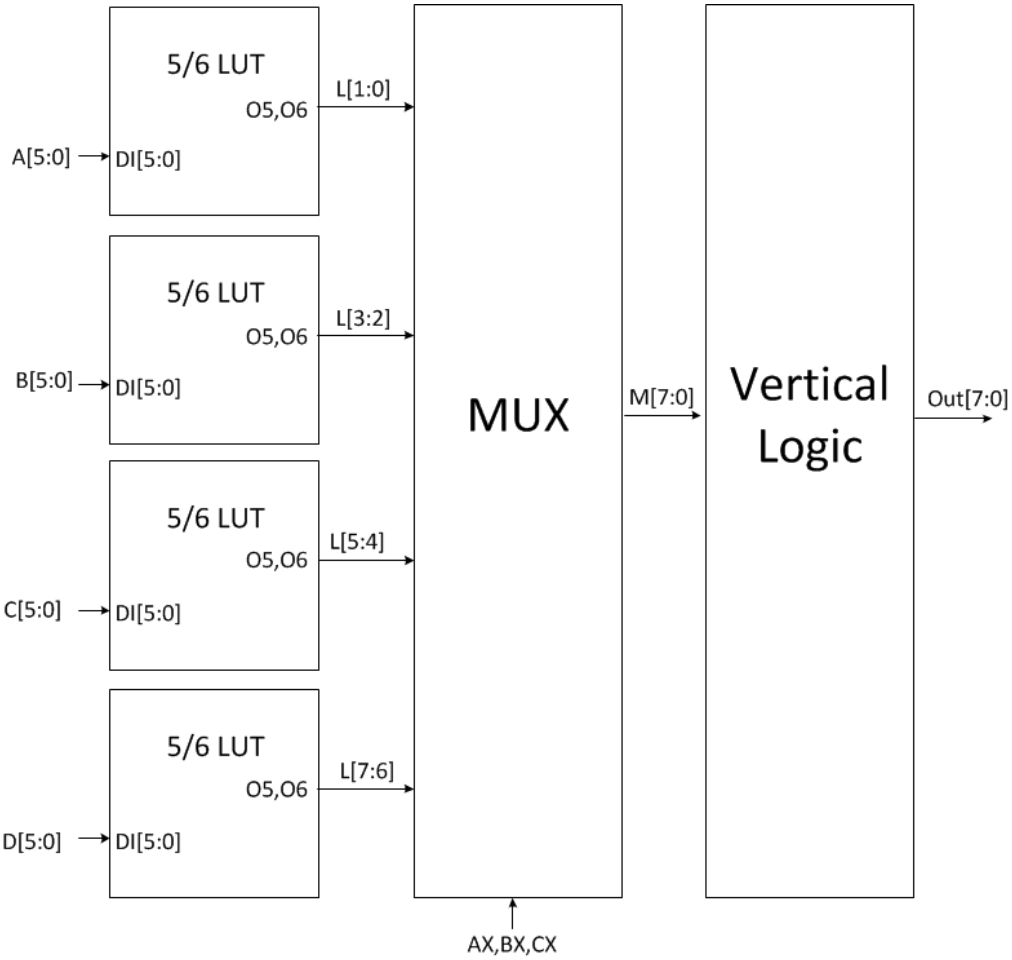


Figure 1. Top level block diagram of a SLICEL.

As shown in Figure 2, a direct implementation of the 5/6 input LUT would use two columns of 32x1 SRAM cells, each implementing a 5-input function (the inputs are DI[4:0]). The outputs of the LUTs can either be both propagated to the output (O5 and O6) or mux-ed using the 6th input DI[5] to implement a 6-input function (O6). To obtain a more reasonable aspect ratio layout, we will instead use four columns of 16x1 SRAM cells, whose outputs are mux-ed twice to generate the same functionality as the ideal cell (Figure 2 on the right).

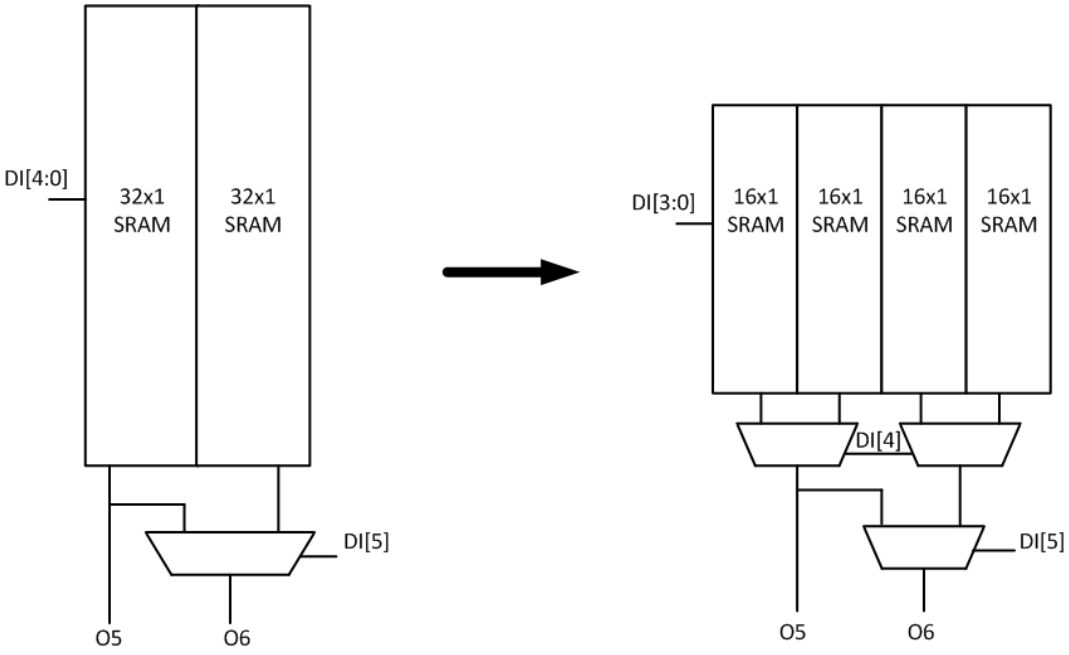


Figure 2: Structure of a 5/6 LUT

In Phase 1, we will assume that all the multiplexers in Figure 2 are included in the big MUX block in Figure 1, and focus our attention on the four columns of 16x1 SRAM cells. Figure 3 shows a more detailed implementation of this cell. It is made of a 4-to-16 decoder and four 16x1 SRAM columns. In addition to these blocks, the subcell also contains circuitry that allows data to be written into the LUTs, and for precharging the bitlines to V_{DD} before the read operation; these circuits are not shown in Figure 3.

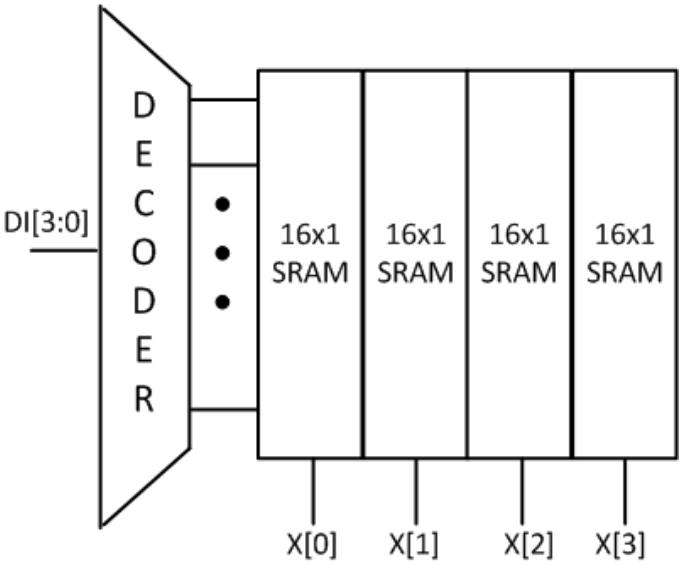


Figure 3: Details of the implementation of the four 16x1 SRAM blocks.

1.2 Implementation and Constraints

The goal of this project is to design a functional, compact, fast, and energy-efficient LUT. The project will be completed in THREE phases by teams of two students. The first two phases of

the project will consist of well-defined tasks (similar to the homeworks), while the final (longest) phase of the project will be much more open-ended.

PHASE 1: Cell Characterization and Decoder Design (due Thursday, Oct. 25, at 5pm)

Cell Characterization:

In the first phase of the project, you are provided with a pre-designed SRAM cell. Characterize the cell stability by using Cadence to obtain an extracted netlist and HSPICE to perform simulations to get the read and write margins.

To obtain the SRAM cell:

- In Cadence, create a library "sram" linked to the gpdk090 90nm technology (see lab 2).
- Now in an x-terminal, go to the directory ~/ee141/sram/ (this assumes that the library you just created is in ~/ee141/sram) and type the following commands:
  cp –R ~ee141/fall12/project/sram_cell/ .
- Go back to Cadence. You should see the SRAM cell design under the cell view sram_cell in your sram library.

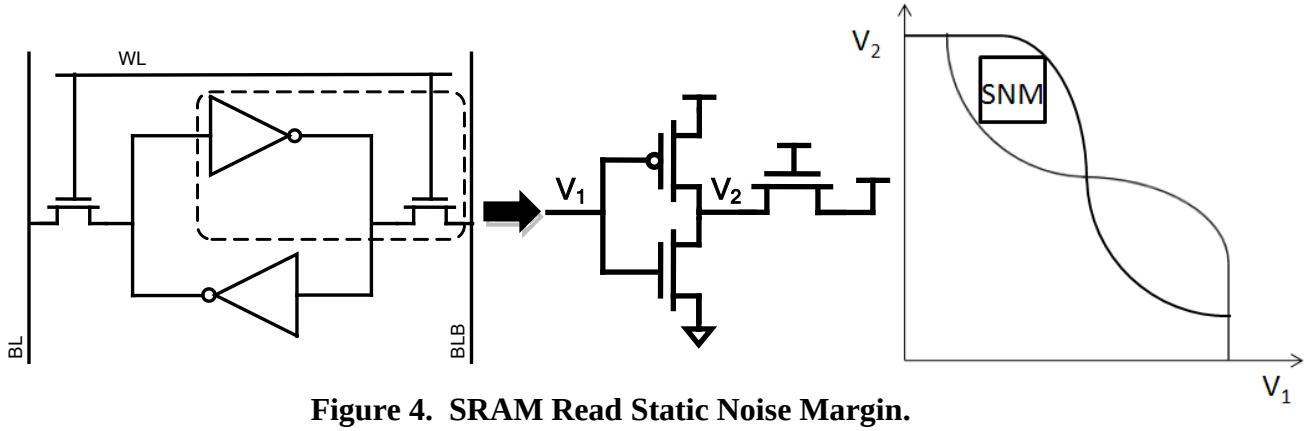


Figure 4. SRAM Read Static Noise Margin.

Recalling that the wordline and bitlines are held at V_{DD} during a read, Figure 4 shows how to extract the read static noise margin (SNM) of the cell. First, the feedback from the cross coupled inverters is broken. Next, the VTC of the "inverter" formed by half of the SRAM cell is found by sweeping V_1 (the inverter's input) from 0 to V_{DD} and measuring V_2 (the inverter's output). This plot is then used to construct the "butterfly plot" that is representative of the two halves of the cell driving each other. The read SNM is the side length of the maximum possible square that can fit inside of the butterfly plot. You do not have to calculate the size of this maximum square, but you should submit the butterfly plot (generated using HSPICE) that graphically indicates the SNM. You should also measure the worst-case voltage rise in the SRAM cell during a read (i.e., the value of V_2 when V_1 is at V_{DD}) and provide that value in your report.

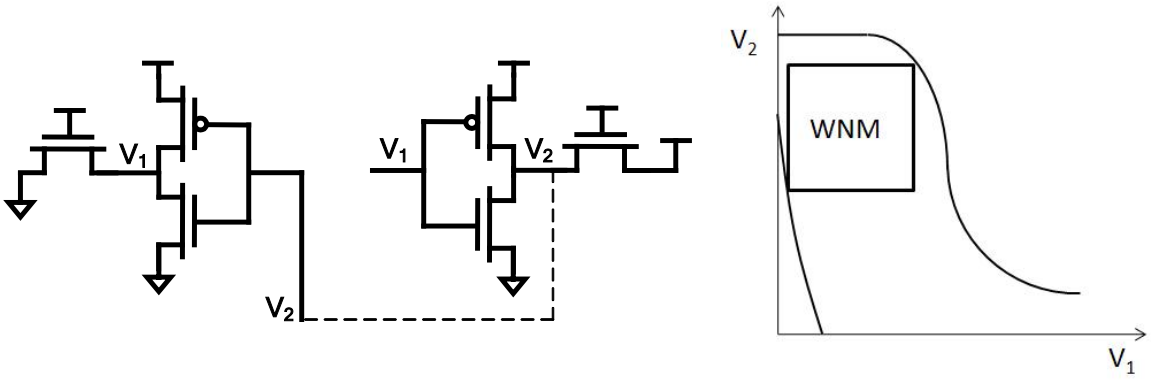


Figure 5. Write Noise Margin.

During a write, VDD is applied to the worldine, and the value to be written into the memory cell is driven onto the bitlines. Thus, Figure 5 shows how to extract the write noise margin (WNM) of the cell. Again, the feedback from the cross coupled inverters is broken, and the VTC of the "inverters" are measured. Note however that in this case, the VTCs of the two halves of the SRAM are no longer the same (since one of the bitlines is driven to 0V, and the other to V_{DD}). These VTCs are used to create a butterfly plot, and the WNM is the side length of the largest square that can fit inside of the butterfly plot. You do not have to calculate the WNM, but you should generate the butterfly plot (again using HSPICE) and graphically indicate the WNM. You should however measure the worst-case cell voltage during a write (which is found from measuring V_1 when V_2 is at 0V).

Layout of SRAM Array:

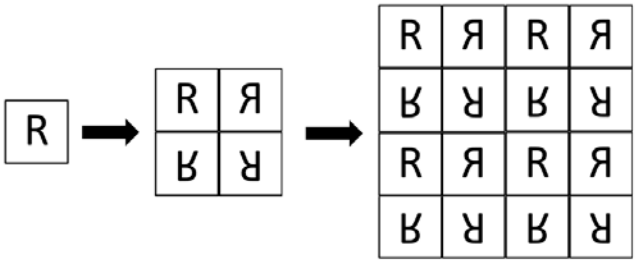
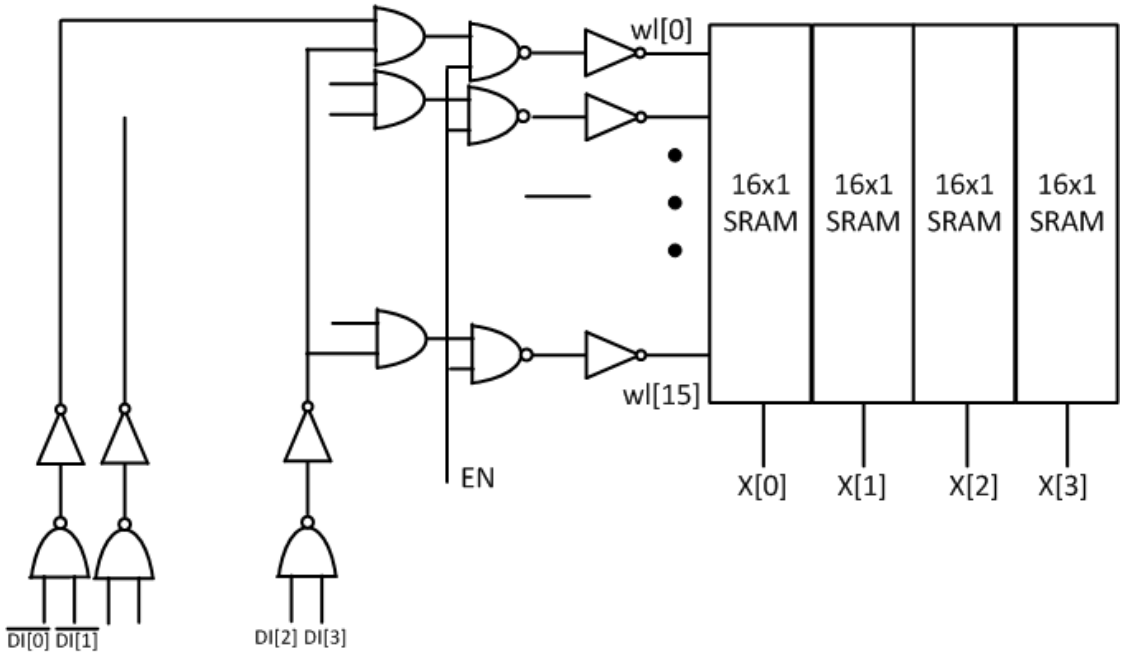


Figure 6. Arraying Procedure

Figure 6 shows how the provided SRAM cell can be arrayed to minimize area. Each adjacent cell is flipped across the X or Y axis. (Note that you may also be able to array the SRAM without flipping across the Y axis.) For phase 1 of the project, you will need to array one row of SRAM cells and use the extracted layout to estimate the capacitive

loading on each wordline (To conduct parasitic extraction of your layout, there will be a tutorial posted under the "Projects" section on the course website).

Decoder Design:



The next stage is to design the 4-to-16 memory decoders. In phase 1 of the project, our goal will be to minimize the delay of the decoder from the address inputs transitioning to the wordline rising. The decoding is performed in two phases: predecoding and then final decoding. The number of input bits to be assigned to the two phases is one of your design choices. The decoder is also provided with an enable signal EN which is active high and enables the decoder outputs. The EN signal is required in order to be able to precharge the bitlines once a read or write has been completed. For phase 1, you can assume that the EN signal will go high at the same time as the address inputs transition.

The predecoder drives the final wordline decoders together with a wire whose length equals the height of the memory array. The logical structure and the number of logic levels of the predecoder and decoder are part of your design choice, e.g., you can exchange NANDs with NORs, or add/remove inverters.

You can assume that both the true and complement versions of the addresses are available to you, but the input loading of each of the signals is constrained to be less than 5fF. The output loading of the decoder is determined from the wordline loading of the SRAM cells and by the wire capacitance. You can ignore the wire resistance, but you should estimate the wire capacitance on both the wordline and the predecode wires by hand (using tables provided in the "Projects" section on the website). In order to get a more precise value for the wordline wire capacitance, you will need to layout and extract one row of the SRAM array, as well as the final stages of the decoder (i.e., the per-row AND gates) that generate the wordline for that row. A few things to note:

- The decoder pitch in layout must match the SRAM cell height.
- Your final decode + SRAM row layout must pass DRC and LVS.
- We have provided a set of standard cell schematics and layout that you can use in your decoder design – these cells are available in the ~ee141/fall12/project/ directory, and can be copied using the same procedure you used to copy the sram cell.
- For this phase of the project, you do not need to construct the entire decoder – you only need to build the schematic for the decoder's critical path.
- Although you will not be laying out the predecoder or its wires, you do need to include the hand estimated values of the predecoder's output wires in your gate sizing.
- Once you have completed the final decode + SRAM row layout, you should extract the layout and then combine the extracted netlist with the rest of the critical path to simulate the delay of your decoder.

In your report, you should include your schematics and layout, your calculation of the worst-case delay from one address input to the wordline rising, and a simulation of this worst-case delay.

Note that before you enter the decoder into Cadence, you should manually design the decoder (on paper) for the minimum delay using the method of logical effort. As part of your report for phase 1, you will need to turn in a one page document that includes your sizing calculations and a diagram of the decoder with all transistor sizes annotated.

Please be sure to attend one of the discussion sessions for tips on how to use Cadence most effectively for this project – these tips will save you a lot of time and debugging, especially in phase 3 of the project.

PHASE 2: MUX-block and adder design (due Friday, November 9, at 5pm)

In the second phase, you will design the MUX stages to select the desired output/outputs from the four 5/6 LUTs as well as the additional vertical logic supporting addition. More details will be provided in the project phase 2 handout.

PHASE 3: Optimization and Assembly (Report due Monday, December 3, at 3:30pm)

In the last phase, you will assemble the entire SLICEL, and have the opportunity to choose a set of optimizations to apply to improve the overall performance, area, and/or power of your design. More details will be provided in the project phase 3 handout.

PROJECT POSTERS are due Wednesday, November 28, time TBA at BWRC.

In each of phases 1 and 2 of the project you will turn in a short report. A longer report is due with phase 3 on December 3. You must be prepared to present your design at BWRC on November 28.

2. Physical and electrical specifications:

2.1. TECHNOLOGY: The design is to be implemented in a 90 nm CMOS process with 5 metal layers. You should only use up to 4 metal layers for the entire design. The SPICE library is located at /home/ff/ee141/MODELS/gpdk090_mos.sp and is the model we have been using all semester (TT_s1v). At a supply voltage of 1.2V, for hand calculations you can assume that L=100nm, C_G = 1.91fF/μm, C_D = 0.77fF/μm, R_{sqn} = 9.43 kΩ/□, and R_{sqp} = 22.32 kΩ/□.

2.2. POWER SUPPLY: You are free to choose any supply voltage and logic swing up to 1.2V. Make sure that you use the appropriate model when you perform any hand analysis.

2.3. PERFORMANCE METRIC: The propagation delay of your memory is defined as the time interval between the 50% point of the inputs and the 50% point of the worst-case output signal. Make sure you pick the worst-case condition and state EXPLICITLY in your report what that condition is!

2.4. POWER: Since everyone's design will be capable of running at different frequencies, we will measure power consumption with a 100MHz clock. Further instructions on how to run simulations to measure power will be provided in Phase 3.

2.5. AREA: The area is defined as the smallest rectangular box that can be drawn around the design. Since the circuit must interface with the rest of the chip, all inputs and outputs must be accessible from the boundary of the layout.

2.6. NAMING CONVENTIONS: (Refer to Figure 1) The inputs of the cell are grouped into four sets of inputs for the 5/6-input LUTS (A-D[5:0]), and three additional inputs to the MUX block (AX, BX, CX). The outputs of the LUTs connected to the MUX block are called *L[7:0]*. The outputs of the MUX connected to the vertical logic are called *M[7:0]*. Finally, the outputs of the cell are called *Out[7:0]*.
(Refer to Figure 3) Wordline and bitline signals refer the SRAM array, and are labeled as *wl*<15:0>, *bl*<3:0>, and *blb* <3:0>. The data entry and the outputs of each SRAM LUT (at the lower hierarchical level) are called *DI*[3:0] and *X[3:0]*, respectively. For the first phase, the complement of *DI*[3:0] that is available is *DIb*[3:0].

2.7. REGISTERS: You don't need to use any registers in this design.

2.8. CLOCKS: You can use as many clocks as you would like for this design. You will need at least one clock to enable/disable the decoder ouputs and precharge the bitlines. Remember that the load capacitance of any clock you use should be included in the power analysis.

2.9. V_{OH}, V_{OL}, NOISE MARGINS: You are free to choose your logic swing in the decoder. The noise margins should be at least 10% of the supply voltage.

2.10. RISE AND FALL TIMES: All input signals have rise and fall times of 100 ps. The rise and fall times of the output signals (10% to 90%) should not exceed 400ps (unless otherwise noted).

2.11. LOAD CAPACITANCE: Each SLICEL output, *Out*[7:0] must drive a 15fF load.

2.12. INPUT CAPACITANCE: For the first phase of the project, the maximum capacitance you can place on each polarity of the SLICEL inputs *A-D[5:0]* (true and complementary) is 5fF. For example, *A*[0] can drive a maximum of 5fF, and *Ab*[0] can also drive up to 5fF.

In addition, the maximum capacitance that you can place on each bit of the SLICEL additional inputs *AX*, *BX*, *CX* connected to the MUX is 10fF.

2.13 EXTRACTION: When performing extraction on your layout, you must use Assura to do parasitic extraction following the directions provided on the website.

3. Simulation

You should always begin your designs by hand analyzing the circuits. Having done this, you should also use HSPICE to simulate the design and prove that it functions correctly. Keep in mind that you will need to determine the input pattern that causes the worst-case propagation delay or energy consumption.

4. Report

The quality of your report is as important as the quality of your design. Be sure to provide all relevant information and eliminate unnecessary material. **Organization, conciseness, and completeness are of paramount importance.** Do not repeat information we already know. Use the templates provided on the web page (Word and PDF formats). Make sure all of your graphs are easily readable, have clearly labeled axes, a meaningful title, and have legends where needed. All figures should have captions that clearly describe what is shown in the figure. Make sure to fill in the cover-page and use the correct units. Turn in the reports for each phase in the homework drop box outside 125 Cory.
In addition, mail an electronic version of your final report and the poster as a Word or PDF file to ee141-project@bwrc.eecs.berkeley.edu. You will also be asked to provide your final netlist.

4.1 Report for Phase 1

The total report should not contain more than five pages. You are not allowed to add any other sheets. The organization of the report should be based on the following outline:

Cover page: Names, summary of simulated SRAM cell parameters, wordline capacitance, and propagation delay.
Page 1: Simulation of the static noise margins.
Page 2: SRAM row layout and schematic.
Page 3: Annotated schematic and the layout of one row of the decoder.
Page 4: Capacitance estimates. Description of the sizing procedure.

Remember, a good report is like a good circuit: it should perform its function (convey information) in the smallest possible area with the least delay and energy (to the reader) possible.

The quality of the report is an important (major) part of the grade!

The total project grade is divided into three parts: the reports from the first two phases, and the final report/poster. The first two parts of the project are worth ~15% of the project grade apiece, while the final report/poster is worth 70%.

For each of the three phases, the grade will be divided as follows:

30% Approach and correctness
30% Results
35% Report
5% Creativity